ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA INFORMÁTICA

APLICACIÓN ANDROID PARA EL INTERNET DE LAS COSAS

ANDROID APP FOR THE INTERNET OF THINGS

Realizado por

Adrián Marín Portillo

Tutorizado por

Mercedes Amor Pinilla

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2016

Fecha defensa:

El Secretario del Tribunal

Resumen:

El presente trabajo aborda la creación de una aplicación para dispositivos Android que, utilizando unas placas de Libelium para recoger el ruido, permite a los usuarios finales ver los niveles de ruido de los lugares donde dichas placas se encuentran.

La información se recoge con unos sensores de ruido integrados en las placas y se mandan a través de una conexión WiFi. La información se manda a un servidor desarrollado para esta tarea por mi compañero Antonio Garcia Jiménez.

La aplicación Android descargará esa información y la hará disponible al usuario a través de un mapa, donde aparecerán marcados como puntos en el mapa las zonas de las que hay información. Los usuarios también podrán enviar audios al servidor, utilizando una grabadora propia de la aplicación que cogerá la media del nivel de ruido (en decibelios), acompañándose de la ubicación del dispositivo, para tratar dicha información.

Palabras claves: android, iot, sensor, mapa, localización, ruido

Abstract:

This proyect addresses the creation of an Android application that, using some Libelium boards to gather the noise, allows the final users to see the noise levels of the places where the boards are located and an estimation of the ocupation of that place.

The information is gathered with a noise sensor integrated within the board, the board sends it via WiFi. This information is sended to a server developed only for this task by my partner Antonio Garcia Jiménez.

The Android application will gather this information and it will make it available in a map. That map will contain the points with the geographical location sended by the server. The users can also send audio files to the server, using a recorder that will take the average noise level in the environment, with the geographical location of the device attached, with the purpose of working with that information.

Keywords: android, iot, sensor, map, location, noise

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción.

1.1. Resumen.

Hoy en día mucha gente dispone de un teléfono móvil, o *smartphone*, en el cual utiliza multitud de aplicaciones, o *apps*, diariamente. La historia de la telefonía móvil daría para crear otro trabajo de gran envergadura, pero es necesario mencionar el fenómeno del *smartphone*. Si bien ya existían sistemas operativos para móviles como Symbian o BlackBerry, no fue hasta el 2007, con el lanzamiento al mercado del primer iPhone, y el 2008, con el lanzamiento del primer teléfono con el sistema operativo Android, el HTC Dream, que se empezó a utilizar este concepto para referirnos a nuestros teléfonos. Desde entonces, la popularidad y las ventas de ambos sistemas ha aumentado como la espuma, llegando a la situación que he mencionado antes, donde entre ambos sistemas suman aproximadamente el 95 % de las ventas de móviles [1], haciendo que otras empresas como Microsoft con su Windows 10 Mobile, BlackBerry con BlackBerry 10 y Cannonical con Ubuntu Touch peleen por el 5 % restante del mercado, provocando en algunos casos su desaparición.

Podemos mencionar muchas categorías en las que podemos clasificar a las aplicaciones que se usan día a día: compras, comunicaciones o incluso ofimática. Pero hay dos categorías, en la que podemos meter a la aplicación sobre la que trata este trabajo, que están en auge: las aplicaciones enfocadas al Internet de las Cosas (*Internet of Things*) y las relacionadas con las Ciudades Inteligentes (*Smart Cities*).

El Internet de las Cosas es un concepto que nació en el MIT (*Massachusetts Institute of Technology*) gracias a Kevin Ashton para referirse a una red global interconectada de dispositivos. Este concepto ha supuesto una nueva revolución al mundo de la tecnología, siendo catalogado como parte de la Web 3.0, que incluye otros conceptos como la web semántica, y supone la introducción de la comunicación entre objetos directamente. Unos objetos con sensores y una conectividad que les permiten realizar realizar tareas de mane-

ra automática y eficiente. El IoT ha podido evolucionar principalmente a factores como el abaratamiento del hardware, las mejoras en miniaturización de componentes y un mejor acceso de cara al público a las tecnologías móviles. No son pocas las organizaciones, tanto públicas como privadas, que están invirtiendo mucho dinero y esfuerzo en esta revolución para poder llevarla hacia los usuarios finales, como los ciudadanos. [2]

Es por los ciudadanos, principalmente, por lo que se crea el concepto de Ciudades Inteligentes. Es un concepto emergente que se refiere a un tipo de desarrollo urbano que es capaz de responder a las necesidades básicas de instituciones, empresas y ciudadanos en el plano económico, social o ambiental. Se considera inteligente a una ciudad cuyas inversiones, principalmente en tecnologías de la información, ayuden principalmente a mejorar la calidad de vida, a un desarrollo económico y ambiental duradero y sostenible y que hagan que los ciudadanos ahorren tiempo en ciertas funciones. En una ciudad inteligente la comunicación entre los principales actores (colectivos, ciudadanos, empresas e instituciones públicas) debe ser fluida y debe haber una participación considerable [3]. El nexo común entre ambas categorías es el mundo de las comunicaciones móviles.

Las redes de comunicaciones móviles se han hecho muy accesibles al público general. Actualmente podemos hacer uso, principalmente, de las redes 3G y 4G, que ofrecen una gran disponibilidad, velocidad e interacción. Gracias a estas tecnologías el uso del *smartphone* ha subido tanto, haciendo que las comunicaciones sean posibles fácilmente. Es por eso que el siguiente paso parece hasta natural: utilizar estas redes para el Internet de las Cosas y las Ciudades Inteligentes.

Si las redes nos han ofrecido tantas posibilidades en el mundo de la telefonía móvil, ¿qué no nos pueden ofrecer para estas dos categorías? Así se abre un nuevo mundo de posibilidades y unas nuevas oportunidades para realizar interesantes proyectos.

1.2. Introducción al proyecto y objetivos.

Este proyecto tiene dos objetivos que se unen en uno solo. Por un lado, realizar una aplicación para dispositivos Android con la cual los usuarios podrán ver en un mapa los niveles de ruidos de ciertos locales de Málaga. También podrá enviar quejas sobre algún otro sitio aportando un audio. Que hacer con esta queja en si no entra dentro del alcance de este proyecto, puesto que ese tratamiento, véase que hacer con la información que se aporta, daría para otro proyecto.

La otra tarea, que será la que alimente principalmente la información sobre el ruido, es el desarrollo de una aplicación para las placas Waspmote, de la empresa Libelium, que recoja el nivel de ruido de donde esté desplegada y envié la información recogida.

Tanto la información que aporten los usuarios como la que recojan los sensores de las

placas será almacenada en un servidor, utilizando una API REST, desarrollado por mi compañero de TFG Antonio García Jiménez, por lo que los detalles de su desarrollo no serán cubiertos en la memoria de mi proyecto.

Si bien este es un proyecto ciertamente ambicioso, las pruebas se realizarán en un pequeño local doméstico al que tengo acceso y en la cafetería de la E.T.S.I.I, para añadir mas variedad al mapa.

1.3. Descripción y justificación de la estructura del documento.

Esta memoria está organizada abordando primero detalles sobre el sistema operativo de Android, como un resumen de su historia, arquitectura y algunos principios sobre el desarrollo de aplicaciones para el sistema. Después va una sección similar para las placas Waspmote, de Libelium, hablando sobre su historia, arquitectura y sensores utilizables.

Una vez expuestos lo necesario sobre ambos mundos, entraré a explicar de forma detallada el desarrollo del proyecto, hablando de las fases por las que ha pasado hasta llegar al entregable definitivo.

1.4. Aclaraciones sobre el anexo del anteproyecto.

En el anexo del anteproyecto había indicado que iba a realizar 260 horas de trabajo, cuando son 296. Aunque el anteproyecto ha sido aprobado sin ninguna pega, me veo en la obligación de incluir una tabla con las horas corregidas.

Fases	Horas
Estudio del funcionamiento de los sensores proporcionados.	15
Búsqueda bibliográfica.	15
Diseño de los programas controladores de los sensores.	50
Diseño de la aplicación Android.	70
Implementación y pruebas de los sensores y la aplicación.	45
Diseño, implementación y pruebas del mapa de ruido.	25
Elaboración de la memoria.	76
Total:	**296**

Tabla 1.1: Tabla de horas corregida

Capítulo 2

El sistema operativo Android.

2.1. Introducción a Android.

Android es un sistema operativo basado en el núcleo Linux. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes o tables; y también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó y que luego, en 2005, adquirió [4].
Android fue presentado en 2007 junto la fundación del *Open Handset Alliance* (un consorcio de compañías de hardware, software y telecomunicaciones) para avanzar en los estándares abiertos de los dispositivos móviles. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008. Actualmente la última versión estable del sistema es la versión 6.0.1, sacada en diciembre de 2015[5].
A día de hoy, Android tiene entre un 80 y un 85 por ciento de la cuota del mercado de teléfonos móviles a lo largo del mundo. En España, a fecha de abril de 2016, Android tiene el 62 % de cuota de mercado, seguido de un 28 % para sistemas con iOS[6].

2.2. Versiones de Android.

Google desarrolla Android a puertas cerradas, y cuando está lista una nueva versión, la empresa libera el código fuente. Raro es encontrar un dispositivo que lleve una ROM con el código tal y como fue lanzado, puesto que las empresas suelen modificarlas para incluir software que, supuestamente, se adapte al hardware del dispositivo.

Desde que la primera versión de Android vio la luz a finales de 2008, ya van 13 versiones importantes, con algunas pequeñas revisiones algunas.

Esas 13 versiones importantes son:

- Android 1.0, que fue la que iba en el HTC Dream.
- Android 1.1, también para HTC Dream, que corregía algunos fallos y cambiaba la Application Programming Interface (API).

- Android 1.5, lanzada en 2010, fue donde se empezó a llamar a las versiones de Android con nombres de postres, siguiendo la tradición de los sistemas basados en Linux de darle un nombre curioso a la versión lanzada , como Debian usando personajes de Toy Story o Ubuntu animales. En concreto se llamó Cupcake.

- Android 1.6, llamada Donut, que incluía mejoras y soluciones a problemas respecto a la cámara y la galería.

- Android 2.0, llamada Eclair, que entre sus mejoras incluía una para el soporte para Bluetooth y el poder soportar HTML 5.

- Android 2.2, llamada Froyo, que fue además donde mas se popularizaron los dispositivos con Android. Incluía varias mejoras importantes, como el soporte para el *tethering*, crear puntos de acceso WiFi o para Adobe Flash.

- Android 2.3, llamada Gingerbread, con soporte para mayores resoluciones, Near Field Communication (NFC) y el cambio del sistema de ficheros de Yet Another Flash File System (YAFFS) a fourth extended filesystem (ext4), más típico en sistemas basados en Linux.

- Android 3.0, llamada Honeycomb, lanzada exclusivamente para tabletas, con mejoras para las mismas tanto en la interfaz como en elementos relacionados, como las notificaciones. También se incluyó la posibilidad de cifrar todos los datos del usuario y el soporte a los procesadores multinúcleo.

- Android 4.0, llamada Ice Cream Sandwich, que fue la última en soportar oficialmente Adobe Flash. Introdujo mejoras como acceder directamente a las aplicaciones desde la pantalla de bloqueo, reconocimiento facial para el desbloqueo o soporte para grabar en 1080p.

- Android 4.1, llamada Jelly Bean, mejoró el rendimiento del sistema, principalmente en lo relacionado con la interfaz, e introdujo las notificaciones expandibles y mejoras en el rendimiento de la cámara.

- Android 4.4, llamada KitKat, introdujo principalmente Android Runtime (ART) como sustituto experimental de la máquina virtual Dalvik en la que se lanza Android.

- Android 5.0, llamada Lollipop, introdujo la interfaz Material Design de Google, un tipo de interfaz *responsive*, y se reemplazó oficialmente Dalvik por ART. También se da soporte a CPU's de 64 bits.

- Android 6.0, llamada Marshmallow, que es la última versión estable y que introdujo mejoras de seguridad y mejoras sobre la gestión de permisos, aparte del soporte para USB Tipo C y el modo de visualización en 4K para aplicaciones.

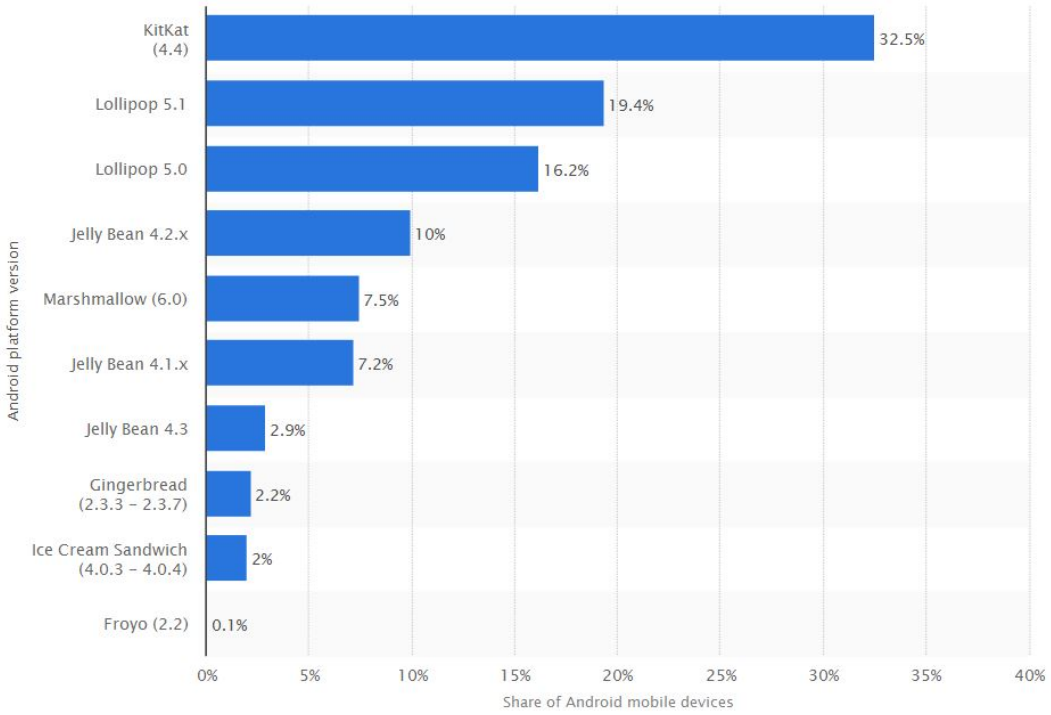

Figura 2.1: Distribución de dispositivos Android según la versión del sistema

La anterior gráfica muestra la distribución de usuarios respecto a la versión de Android que utiliza [7] . Para este proyecto se ha tenido en cuenta la distribución de los dispositivos Android, teniendo en cuenta también las herramientas disponibles a nivel de desarrollador.

2.3. Arquitectura de Android.

Android se basa en un *kernel* de Linux, sobre el cual hay ciertas librerías y API's escritas en C, un *framework* de aplicaciones y un conjunto de aplicaciones en la capa de más alto nivel.

Linux Kernel

Hay que aclarar primero que Android está basado en Linux pero no es una distribución de Linux, sino que los desarrolladores pueden modificar el *kernel* para adaptarlo a las necesidades propias de Android, construyendo un sistema totalmente distinto. Este núcleo incluye los drivers necesarios para el funcionamiento del dispositivo.

Libraries

Son librerías escritas en C/C++ que hacen operaciones de forma mas sencilla para el programador y ayudan a mejorar el rendimiento. Un ejemplo serían las librerías de SQLite o OpenGL.

Android Runtime

Las aplicaciones Android se ejecutan sobre una máquina virtual. Primero fue Dalvik, utilizando compilación Just In Time (JIT) de Java a Java *bytecode*, y luego el resultado se traduce a un fichero con formato .dex para ser entendido por Dalvik. Desde Android 5.0 se utiliza *Android Runtime* (ART), que usa compilación *ahead of time* (AOT) y otras técnicas para mejorar la velocidad y el rendimiento.

Application Framework

Aquí se encuentran recursos que utilizan los desarrolladores para realizar las aplicaciones. Se puede destacar el *Activity Manager*, para la gestión del ciclo de vida de las actividades, los *Content Providers*, para compartir datos entre aplicaciones distintas, o el *Location Manager*, para tareas relacionadas con el Global Positioning System (GPS).

Applications

La capa superior incluye las aplicaciones básicas que incluye el sistema, como el gestor de contactos, el navegador, etc

Figura 2.2: Capas de la arquitectura de Android

2.4. Desarrollo de aplicaciones para Android.

Las aplicaciones Android están escritas, principalmente, en Java, empleando para ello el *Software Development Kit* (SDK) que proporciona el *framework* necesario: librerías, depuradores, emuladores, códigos de ejemplo, etc. El SDK de Android está disponible tanto para Windows, Mac OS X y GNU/Linux.

Otras posibilidades destacables para realizar aplicaciones para Android son:

- Kotlin: Es un lenguaje desarrollado por JetBrains que corre sobre cualquier máquina virtual de Java. Es compatible 100 % con Android, puede ser utilizado en un proyecto en el cual se utilice Java, y nos da ciertas ventajas que aún no están en Java, como las funciones lambda, y otras que no están ni en versiones mas actuales de Java, como un manejo mas seguro de los objetos null. Es un lenguaje a tener en cuenta en el futuro, puesto que Google lo está promocionando mucho[8].

- Xamarin: Creado por los ingenieros que desarrollaron el compilador Mono, es un software bajo el cual se pueden desarrollar aplicaciones móviles para múltiples plataformas (Android, iOS y Windows Mobile) utilizando C#

- Corona: Un SDK basado en el lenguaje LUA pensado para realizar aplicaciones de manera mas sencilla que en Java.

- NDK: Utilizando la JNI (*Java Native Interface*), los desarrolladores pueden utilizar código escrito en C/C++, y haciendo que se aproveche mas el procesador del dispositivo.

Anteriormente se usaba principalmente Eclipse, pero Google descontinuó su soporte en 2015, animando a los desarrolladores a utilizar el nuevo entorno oficial, Android Studio. Android Studio está basado en IntelliJ IDEA, un entorno desarrollado por JetBrains (que tiene Integrated Development Environment (IDE)'s para otros lenguajes como Java, PHP, Python, etc). Fue presentado en mayo de 2013 en una Google I/O Conference y se lanzó públicamente a finales de 2014. Recientemente ha sido lanzada la versión 2.1, mejorando los tiempos de compilación y añadiendo mejoras como *instant run*, para probar código de manera más rápida. También está disponible para Mac OS X, GNU/Linux y Windows[9]. Sus características principales son:

- Renderización en tiempo real.

- Construcción del archivo .apk usando Gradle, sistema heredero de Ant y Maven.

- Posibilidad de crear *Build Variants*, versiones/configuraciones distintas de la aplicación según el dispositivo para el cual se lance u otras reglas.

- Integración mas fácil de *Unit Testing*.

- Optimización del proyecto mediante el uso de Android Lint, que estudia el código y te destaca que se debería arreglar o cambiar.
- Herramientas para un mejor desarrollo de interfaces.

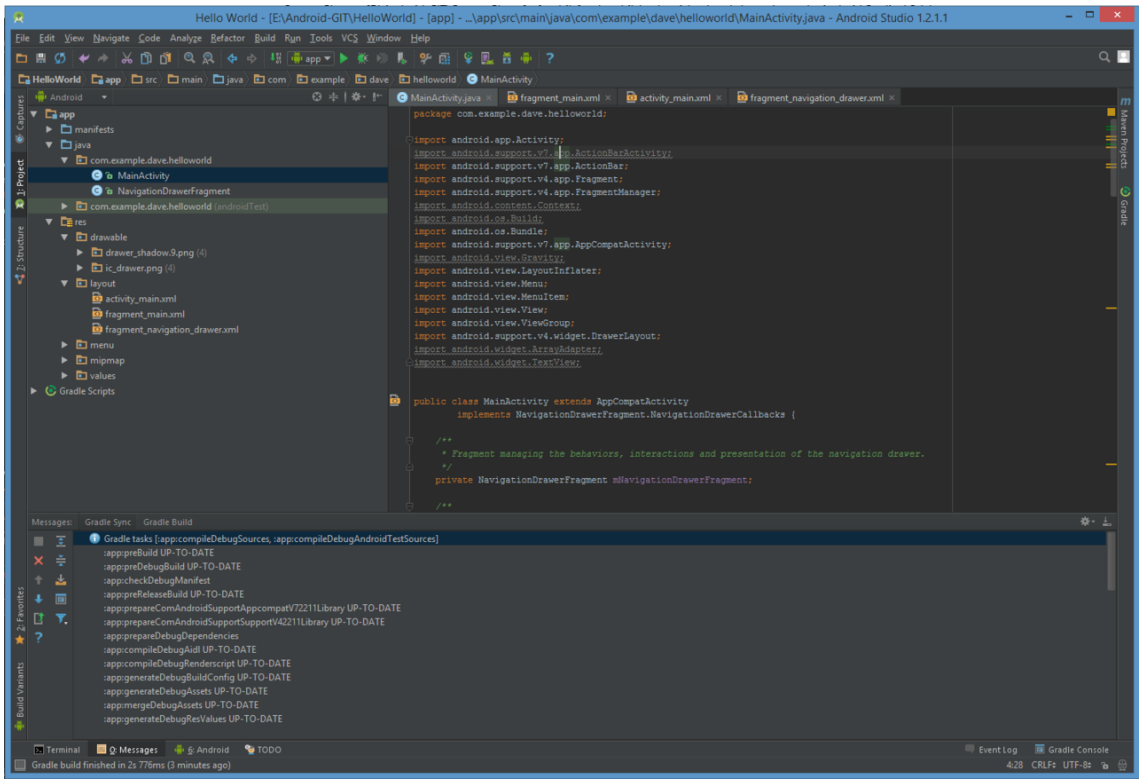

Figura 2.3: Pantalla de Android Studio

El código de una aplicación Android se divide en tres partes principales [10]

- Las clases Java, que podrán ser de los siguientes tipos:
  - Clase típica en Java, pudiendo utilizarse interfaces, que sean clases abstractas, etc.
  - Actividades, que manejan la lógica de la visualización y organizan el funcionamiento general de la aplicación.
  - Servicios, que son procesos en *background*.
  - *Content Provider*, para compartir información a través de las aplicaciones.
  - Etcétera, etcétera.
- El archivo *manifest* que incluye la configuración básica de la aplicación: la lista de actividades, los permisos que se necesitan, etc.
- Los archivos eXtensible Markup Language (XML) para distintas tareas:
  - Almacenar valores de cadenas de texto (que pueden tener distintos valores según el idioma del sistema) o estilos de diseños.

- Claves para API's externas.
- Diseñar las interfaces/*layouts* de usuario, pudiendo realizarse de distinta forma según la resolución del dispositivo donde se instale.

2.5. Publicación de aplicaciones para Android.

La manera oficial para distribuir las aplicaciones Android es utilizando el Google Play Store, anteriormente conocido como Android Market. Se trata de una plataforma de distribución digital donde los usuarios pueden acceder tanto a aplicaciones como películas, música, revistas, juegos, etc. Para el primer trimestre de 2016 se ha llegado a los casi 2 millones de aplicaciones publicadas[11].
Existen tanto las aplicaciones gratuitas como las de pago, y en ambas se acepta el uso de los micro-pagos. Para poder subir aplicaciones como desarrollador solo hay que hacer un pequeño pago único de 25 dolares. Google no hace ningún tipo de revisión a las aplicaciones que subimos, a diferencia de Apple que impone revisiones muy complicadas haciendo que haya menos aplicaciones en su tienda, aunque si revisa luego posibles denuncias de falsificación o infracciones del copyright[12].

Hay otras tiendas para poder descargar aplicaciones, aunque no mencionaré ninguna aquí, ya que son muchos los casos en los que estas otras tiendas han sido utilizadas para distribuir software falso o con virus. También pueden ser instaladas manualmente, descargando el archivo .apk, desactivando antes una opción de seguridad en los ajustes del dispositivo.

Capítulo 3

Placa Waspmote.

3.1. Introducción a Waspmote.

Waspmote es una plataforma de sensores *wireless* de código abierto pensada para que el desarrollador tenga el control total de una o varias motas autónomas que sigan una filosofía en la que los componentes necesarios sean sencillos de añadir a la mota. Se pueden preparar redes inalámbricas de sensores bastante fáciles y escalables con esta plataforma. Estas motas fueron desarrolladas por la empresa Libelium, de Zaragoza, que fue fundada en 2006 por Alicia Asín y David Gascón. Se considera una empresa pionera en el mundo del Internet de las Cosas, y actualmente trabajan con muchas empresas y organismos públicos para llevar sus productos a todas partes del mundo[13].

Waspmote empezó como un proyecto derivado de Arduino, por eso utilizan un compilador y *drivers* similares y ambos tienen un entorno de desarrollo similar.

3.2. Características técnicas y arquitectura.

La Waspmote Pro v1.1, que es la que utilizo en este proyecto y una de las últimas a la venta, tiene las siguientes características técnicas:

- Microprocesador Atmega1281 a 8 Mhz.
- 128Kb de memoria Flash, 8Kb de Static Random Access Memory (SRAM) y 4Kb de Electrically Erasable Programmable Read-Only Memory (EEPROM).
- Dimensiones de 73.5x51x13 mm y un peso de 20 gramos.
- Admite desde -20 a 60 grados centígrados.
- Incluye un reloj Real Time Clock (RTC) (*Real Time Clock*) a 32 Khz.

- Consume encendido 15mA, en modo *sleep* 55μA, en modo *deep sleep* 55μA y en modo hibernación 0.06μA.

- Entradas/salidas:

  - 7 entradas análogas.
  - 8 entradas/salidas digitales.
  - 2 UART (*Universal Asynchronous Receiver-Transmitter*).
  - 1 L2C (*Inter-Integrated Circuit*).
  - 1 SPI (*Serial Peripheral Interface*).
  - 1 entrada USB.
  - Varios *sockets* específicos para ciertos sensores (temperatura, ruido, etc).

- Voltaje de la batería de 3.3V a 4.2V

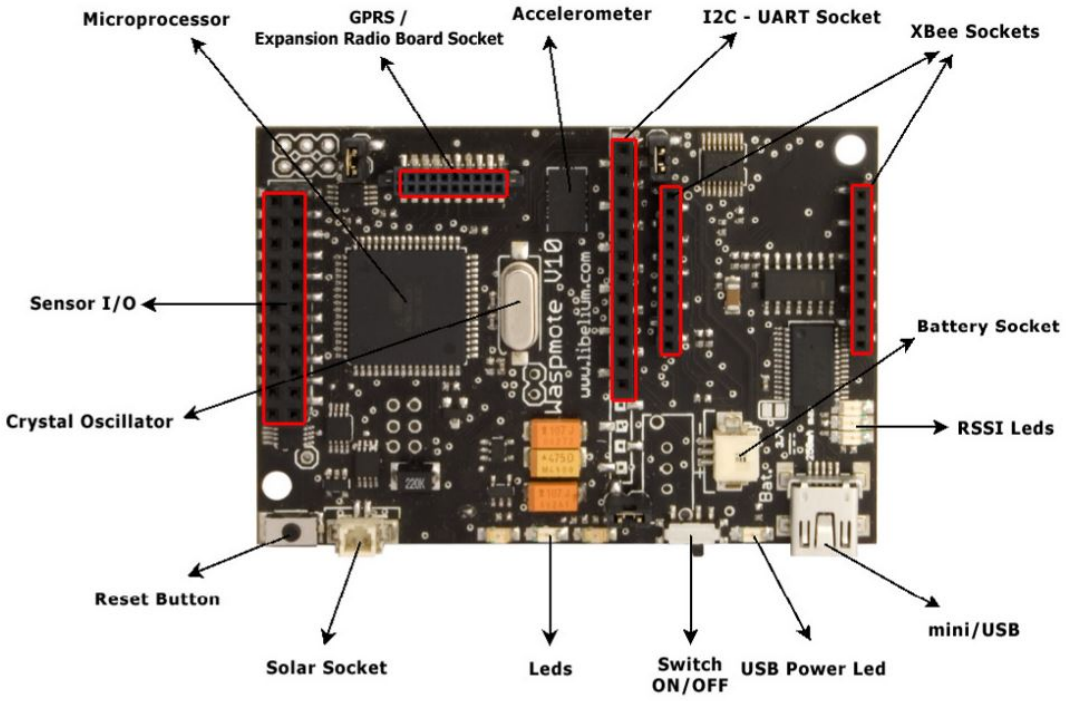

Figura 3.1: Parte delantera de una mota

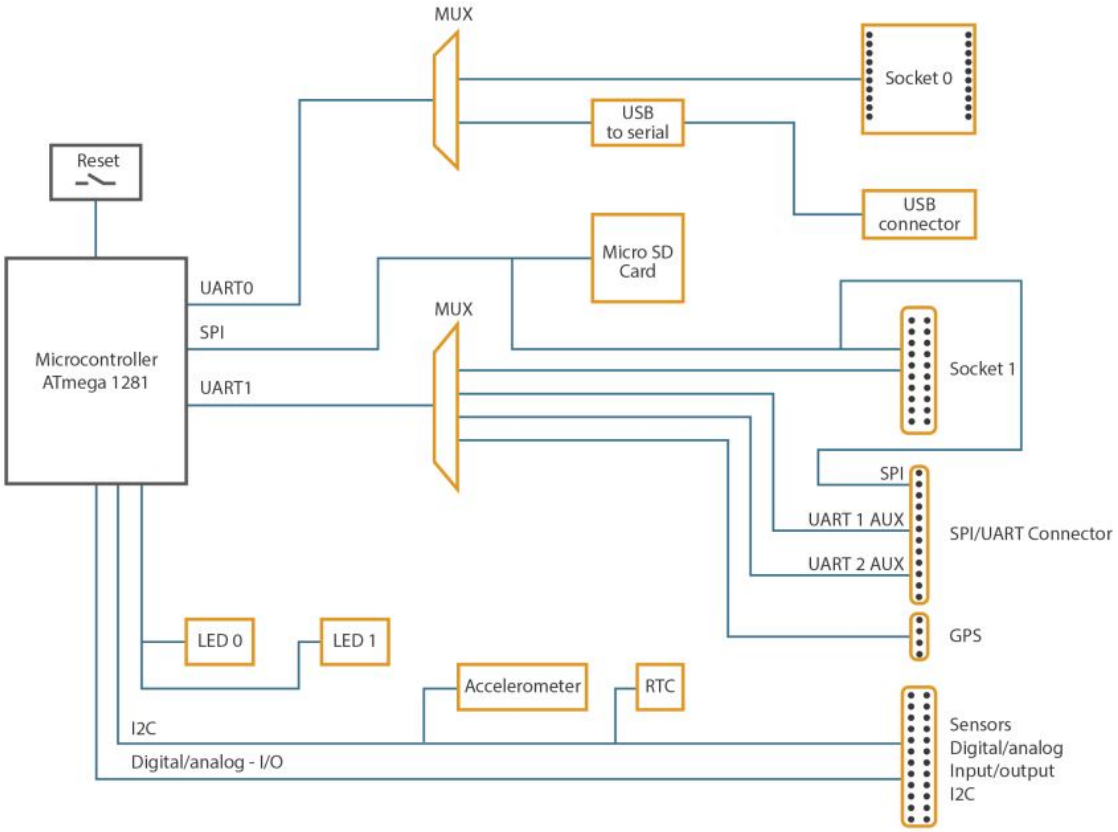

Figura 3.2: Diagrama del flujo de datos de una mota

Waspmote está pensado para tener una arquitectura modular, en la que cada módulo solo sea instalado si es necesario, ahorrando en costes y complejidad. Un módulo integrable de Waspmote es un circuito diseñado para funcionar con estas motas al instante, tan sólo hay que insertarlo en su respectivo hueco, y mediante la API que ofrece, que debe ser importada en el *sketch*, se pueden utilizar sus funciones de manera sencilla. Los módulos integrables principales se describen en los siguientes apartados

3.3. Sensores.

Los sensores se utilizan a través de las placas de sensores que ofrece Libelium. Estas placas permiten a los desarrolladores acceder a diversos tipos de sensores fácilmente, sin tener que preocuparse por problemas como los referentes al cableado, mediante la API que ofrece cada una. Gracias a estas placas podemos medir las variaciones en una gran variedad de medidas posibles. En las motas se pueden integrar las siguientes placas de sensores:

- Gases, desde los mas propios de procesos industriales como de polución o incluso para detectar incendios. Ejemplos: CO, CO2, O3
- Gases PRO, que pueden detectar partículas de polvo y gases mas 'complejos' para los sensores anteriores. Ejemplos: CH4, HCl

- De eventos, como cambios en la presión, temperatura, efecto Hall, vibraciones, etc.
- *Smart Water* para propiedades relacionadas con el agua, como el pH, los nitratos que tiene, la salinidad, etc.
- *Smart Cities*, para recoger información sobre el ruido, los ultrasonidos, la temperatura, etc.
- *Smart Parking*, para la gestión de aparcamientos. Para la agricultura, pudiendo detectar la temperatura, la radiación ultravioleta, la humedad, etc.

Aunque se mencione varias veces, por ejemplo, la temperatura en lo que pueden recoger los sensores, estos sensores no son intercambiables entre si. Es decir, un sensor de la categoría de eventos para recoger la temperatura no es válido para la agricultura, puesto que está expuesto a unas condiciones totalmente distintas que determinan su calibración y su medición en un entorno ambiental concreto.

Figura 3.3: Mota con sensor de presencia

3.4. Interfaces de comunicaciones.

Las motas pueden aprovechar de los siguientes protocolos importantes usando sus respectivos módulos:

- 802.15.4/ZigBee, con cifrado AES 128b y pudiendo usar las topologías de árbol, malla y P2P.

- LoRaWan, una especificación de LPWAN (*Low Power Wide Area Network*).
- WiFi/802.11b/g, a 2.4 Ghz. Soporta cifrado Wired Equivalent Privacy (WEP), Wi-Fi Protected Access (WPA) y WPA2.
- RFID/NFC
- 3G + GPS
- Bluetooth *Low Energy* 4.0 y Bluetooth PRO.
- *Expansion Radio Board*, para hacer combinaciones entres los protocolos mencionados.

3.5. Desarrollo de programas para Waspmote.

Al ser un proyecto derivado de Arduino, utiliza un IDE similar a este, teniendo que descargar y poner en el mismo directorio la API actual de las motas. En este momento van por la revisión número 21, introduciendo algunos arreglos sobre métodos.

Figura 3.4: Interfaz del entorno de desarrollo.

Los programas son llamados *sketch*, se escriben en un lenguaje similar a C, con ciertas restricciones. Primero se compilan y luego son flasheados en la memoria de la mota, donde empiezan a funcionar al momento. Podemos ver la salida por pantalla utilizando el puerto serial.
Una de las restricciones encontradas en el desarrollo del proyecto fue que la función para concatenar cadenas que utilizaba, *sprintf*, no valía para concatenar con un número de punto flotante debido a que la versión del compilador era anticuada. Este problema fue subsanado desarrollando una función propia para hacer la concatenación.

Capítulo 4

Desarrollo e implementación del proyecto.

Primero voy a describir los pasos para el despliegue de la mota Waspmote, del funcionamiento del *sketch* y los problemas y sucesos relacionados con el desarrollo.

4.1. *Sketch* para Waspmote.

4.1.1. Explicación del funcionamiento del sketch.

Todos los *sketchs* de Waspmote, como los de Arduino, se dividen en dos funciones principales: la función *setup* y la función *loop*, ambas de tipo void. En la cabecera del fichero, antes de estas funciones, van los include (importar librerías, como en C) para el funcionamiento del programa. Aquí se importan librerías como las que controlan los sensores, por ejemplo.

La función *setup* prepara todos lo necesario para el correcto funcionamiento de la mota, es decir, inicializa variables, enciende sensores, prepara el reloj interno, etc. La mayor parte de problemas relacionados con el código del problema suelen estar aquí.

La función *loop* se ejecuta siempre que tenga batería la mota. Aquí va todo el código necesario para las acciones que debe ejecutar la mota. Se pueden crear otros métodos aparte para mejor lectura del código, pero esos métodos se llamaran aquí. Para ahorrar batería, las motas pueden ser programadas para entrar programáticamente en distintos modos de ahorro que voy a describir a continuación[14]:

Modo	Consumo	Ciclo de suspensión	Llamadas válidas
Encendido	15mA	-	Interrupciones asíncronas y síncronas
Sleep	55μA	32ms a minutos/horas/días	Reset e interrupciones asíncronas
Deep sleep	55μA	1s a minutos/horas/días	Interrupción de reloj y asíncronas
Hibernación	0.06μA	1s a minutos/horas/días	Interrupción de reloj.

Tabla 4.1: Modos de energía

- Encendido: Todas las funciones están operativas
- *Sleep*: Algunas funciones de algunos módulos son paradas, siendo despertadas normalmente por eventos.
- *Deep sleep*: Igual que *Sleep* pero aumentando el tiempo y pudiendo ser despertado por interrupciones programadas de reloj.
- Hibernación: Todas las funciones de todos los módulos son paradas y serán despertadas por eventos.

Estos modos operacionales son muy útiles para ahorrar batería, aumentando el ciclo de vida de nuestra mota y hace que no dependa durante más tiempo de un operario humano, pero la elección del modo a usar no puede ser a la ligera. Algunos modos dan conflictos con ciertos módulos, por lo tanto es recomendable investigar antes de decidirse por un modo a la hora de programar si es compatible con los módulos que usamos, puesto que el modo de ahorro es negociable, pero el uso de un módulo u otro rara vez lo va a ser.

En este proyecto se usará el modo *sleep*, puesto que, según la documentación, conserva todo el estado de nuestras variables, mientras que el modo de hibernación no lo hace, y es el mas sencillo de programar.
El funcionamiento del *sketch* programado es sencillo: tras iniciarse el programa, la mota encenderá el micrófono, recogerá el valor y lo enviará, utilizando una conexión WiFi, al servidor API de mi compañero Antonio. Tras eso entrará en modo *deep sleep* durante 5-10 minutos, para conservar batería y descartar medidas muy sucesivas.
La documentación del sensor nos aporta hasta cuantos decibelios soporta y que podemos asociar a ese nivel de ruido:

Sonido	dBA
Umbral de audición	0
Habitación vacía	30
Conversación normal	60-70
Mucho tráfico	90
Umbral del dolor	130
Motor de avión cerca	140

Tabla 4.2: Relación ruido-decibelios

Para la parte de enviar los datos, debemos usar la librería WiFi para la mota. Para conectarnos con un punto de acceso hay que introducir en el código tanto el ESSID del punto como su contraseña, aunque luego en el intercambio irá cifrada esta información. Una peculiaridad que tiene es que, a pesar de que en el fondo lo que vamos a hacer es un POST a un servidor, la librería no lo llama así, sino que hay que usar la función Wifi.getUrl(), pasando como argumentos si la petición será usando DNS o la IP directamente, el nombre de dominio o la IP y el método a ejecutar, en nuestro caso POST.

Ejecutaremos un POST con el valor y la localización del sensor, ambos valores concatenados. Al no contar con un módulo de GPS la localización estará incluida en una variable por defecto del *sketch*.

A la hora de estudiar la documentación de las motas se consideró para las comunicaciones el uso de Bluetooth. Concretamente, utilizar el módulo de Bluetooth de una mota para enviar la información recogida a un teléfono con la aplicación para que este teléfono actuase a modo de pasarela entre la mota y el servidor. Aunque parece una buena idea para hacer más fácil el despliegue, la documentación del módulo explica que el módulo no puede enviar dicha información, aparte de la complicación extra que conlleva programar la mota de esta manera. Otra posibilidad fue utilizar el router Meshlium, pero no se necesitan sus funcionalidades extra, por lo que usaré un router normal.

4.1.2. Despliegue en el entorno de pruebas.

Para hacer las pruebas para esta memoria he probado la mota en 2 lugares: una habitación de un local al que he tenido acceso y la cafetería de la Escuela Técnica Superior de Ingeniería Informática.

4.1.3. Código del *sketch*.

```
// Libreria para el microfono
#include <WaspSensorCities.h>
//Libreria para la conectividad WiFi
#include <WaspWIFI.h>
```

```
//Libreria para modificar strings
#include<string.h>
#include<stdio.h>
//Socket para la conexion WiFi
uint8_t socket=SOCKET0;
// Variable para guardar el valor del microfono
float value;
char value_message[100];
//Direccion IP y puertos
#define IP_ADDRESS "http://150.214.108.91"
#define REMOTE_PORT 8000
// Configuracion del punto de acceso
#define ESSID "test_tfg"
#define AUTHKEY "850e86eca"
// Timeout de los mensajes
#define TIMEOUT 10000
//Coordenadas hard-codeadas de una de las pruebas
char gps []= "36.7102897,-4.4668383,17z";
//Respuesta de la peticion
uint8_t status;

void setup() {
  USB.begin();
  delay(100);
  //Enciende el sensor
  SensorCities.setBoardMode(SENS_ON);
  //Enciende el reloj
  RTC.ON();
  //Activa la conexion Wifi
  wifi_setup();
  value = 30; //Umbral de ruido. Inicializamos por si acaso.
}


void loop() {

  // 1. Enciende el modulo WiFi
  SensorCities.setSensorMode(SENS_ON, SENS_CITIES_AUDIO);
  delay(2000);
  value = SensorCities.readValue(SENS_CITIES_AUDIO);
  ftoa(value,value_message,3);
  strcat(value_message,",");
```

```
  strcat(value_message,gps);
  USB.print("sentence:");
  USB.println(value_message);
  if (WIFI.join(ESSID))  {
    USB.println(F("Conexion con el AP establecida"));
    sprintf(sentence,"POST?",value_message);
    USB.print("sentence:");
    USB.println(sentence);
    status = WIFI.getURL(IP,IP_ADDRESS,sentence);
    if(status==200) {
      USB.println(F("\nPeticion OK."));
      USB.println(WIFI.answer);
    }
    else{
      USB.println(F("\nERROR PETICION"));
    }
  }
  else {
    USB.println(F("Conexion fallida"));
  }
  SensorCities.setSensorMode(SENS_OFF, SENS_CITIES_AUDIO);
  /*
  Modo sleep durante 1 minuto.
  La placa no admite valores superiores a 8 segundos
  asi que para dormir un minuto, 60 segundos,
  debe dormir 15 veces durante 4 segundos
  */
  for(int i=0; i<16;i++) {
    PWR.sleep(WTD_4S, ALL_OFF);
  }
}

/*************************************
 *
 *  wifi_setup
 *  funcion para configurar la conexion WiFi
 *
 *************************************/

void wifi_setup() {
  // Encender el modulo WiFi en el socket deseado
  if( WIFI.ON(socket) == 1 ){
```

```
    USB.println(F("WiFi encendido"));
    // 1. Configura el protocolo a seguir.
    WIFI.setConnectionOptions(HTTP|CLIENT_SERVER);
    // 2. Configura como obtener la IP.
    WIFI.setDHCPoptions(DHCP_ON);
    // 3. Configura como conectarte al AP
    WIFI.setJoinMode(MANUAL);
    // 4. Configurar modo de autenticacion.
    WIFI.setAuthKey(WPA1,AUTHKEY);
    // 5. Guardar cambios
    WIFI.storeData();
  }
  else {
    USB.println(F("WiFi no funcionando"));
  }

}
/*************************************
 *
 *  ftoa
 *  funcion para convertir de double a char,
 *      con una cierta precision
 *************************************/
char *ftoa(double f,char * a, int precision) {
  long p[] = {
    0,10,100,1000,10000,100000,1000000,10000000,100000000
};
  char *ret = a;
  long heiltal = (long)f;
  itoa(heiltal, a, 10);
  while (*a != '\0') a++;
  *a++ = '.';
  long desimal = abs((long)((f - heiltal) * p[precision]));
  itoa(desimal, a, 10);
  return ret;
}
```

4.2. Aplicación para dispositivos Android.

4.2.1. Descripción del proyecto y diversas decisiones.

Para hablar sobre el funcionamiento de la aplicación antes hay que comentar ciertos aspectos sobre los componentes y partes de Android que he utilizado en el desarrollo y que hay que explicar:

- *Fragment*: Introducidos en Android 3.0 y pensados para su uso en tabletas, permiten representar en una parte de la pantalla cierta información, pudiendo utilizar la otra parte para otra tarea. En este caso, he utilizado un *fragment* específico llamado *SupportMapFragment* que es de la API de Google para dispositivos Android, para integrar fácilmente un mapa dentro de la propia aplicación y poder manipularlo según mis necesidades.

- Volley: Es una librería desarrollada por Google para realizar las peticiones Hypertext Transfer Protocol (HTTP) de manera más sencilla y rápida, permitiendo múltiples conexiones de red concurrentes sin tener que complicarnos con los aspectos de la concurrencia. Anteriormente se utilizaba para las peticiones la librería de Apache, pero ya no se incluye en Android desde el nivel de API 23 (Android 6.0) y se consideró *deprecated* en el nivel de API 22 (Android 5.1). Para utilizarlo se puede incluir una linea en el archivo de Gradle o se puede incluir como .jar en las librerías.

- *AsyncTask*: Se utiliza esta clase, concretamente creando una clase que herede de ella, para realizar operaciones de background complicadas que puedan tardar, evitando que se realicen en el hilo/thread de la interfaz gráfica. Esto se hace así debido a que estas operaciones pueden congelar dicho hilo, y si lo hacen más de 5 segundos el programa lanza una excepción del tipo ANR (*Android Not Responding*).

- *LocationManager*: Un recurso que pertenece dentro del *stack* de Android al *framework* de aplicaciones y que gestiona los sistemas de localización. Concretamente en este proyecto se utiliza para la localización GPS.

- *Handler*: Utilizado para mandar y recibir procesos entre aplicaciones y/o clases. En este proyecto se utiliza junto a la clase que toma los valores de ruido para manejar dicha información.

La aplicación tiene 3 pantallas principales:

- Inicio: En esta pantalla, que es la que se carga al iniciar la aplicación, se elige la acción a lanzar, que puede ser consultar el mapa con los niveles de ruidos, mandar un audio propio, una pantalla con información sobre la aplicación o compartir un mensaje en redes sociales. La actividad que gestiona esta pantalla es *MainActivity*.

Figura 4.1: Vista previa de la actividad de inicio.

- Niveles de ruido: En esta pantalla podemos ver, gracias a la API de Google Maps, un mapa, que está controlado por un *fragment*, en el cual se pueden ver los marcadores de los puntos de los cuales el servidor tiene información. Esta pantalla está controlada por la actividad *NoiseActivity* y para poder entrar en ella el usuario debe activar el GPS, ya que se necesita dentro (al intentar entrar sin tener activo el GPS se muestra un aviso al usuario para que lo active).

Figura 4.2: Vista previa de la actividad de la grabadora.

- Enviar prueba de ruido: Aquí el usuario puede grabar un ruido que considere que debe enviar a los servidores. Se crea dentro un grabador de sonidos, en vez de reutilizar el que tiene incorporado Android, porque así, al salir de la pantalla, se puede enviar el audio al servidor fácilmente (el grabador nativo de Android daba problemas respecto a este tema). La actividad controladora de esta pantalla es *RecordActivity* y también es necesario que se active el GPS, funcionando en este sentido igual que la actividad anterior.

Los otros dos botones que aparecen en la aplicación de inicio corresponden a un botón en el que se repite de manera mas sencilla esta información, de cara al fácil entendimiento del usuario final, y un botón para compartir un mensaje en las redes sociales sobre la

aplicación o para mandar un correo.

Un aspecto principal de todo desarrollo en Android es escoger que nivel de API es el mínimo con el cual puede funcionar. Para compilar se recomienda utilizar las herramientas incluidas en las APIs mas modernas (concretamente se utilizan las librerías de *support*), pero siempre hay que indicar un nivel mínimo de API. Esto es así debido a como funcionan los niveles: cuando vamos subiendo de nivel se añaden funcionalidades nuevas y otras son sustituidas por nuevas implementaciones. Otro aspecto fundamental a la hora de escoger nivel es el porcentaje de dispositivos que hay en uso que tengan mínimo esa API. Hemos podido ver en la Figura 1 el porcentaje de dispositivos que utilizan cada versión de Android, y por tanto la API que usan. Es por ello que, teniendo en cuenta la distribución de su uso y las ventajas que se añaden, he escogido para este desarrollo utilizar como nivel mínimo la API 16, que se corresponde con Android 4.1, pudiendo abarcar casi el 95 % de los dispositivos que hay en uso actualmente.

4.2.2. Estructura de la aplicación.

Las partes destacables de todo proyecto Android son:

- Archivo *manifest*.
- Los componentes que controlan la vista y lógica de la aplicación.
- Los recursos, como cadenas de texto, imágenes, etcétera.
- Archivos relacionados con Gradle.

El núcleo del funcionamiento de la aplicación es el archivo del *manifest*, que es un archivo xml donde se incluye, siguiendo la estructura típica de este tipo de archivos:

- Los permisos que se solicitan a la hora de instalar (API menor o igual que la 22) o en tiempo de ejecución (API superior que la 22). Si en una actualización de la aplicación se añade una nueva petición de permisos, se avisa al usuario antes de ejecutar la actualización. Solo se deben incluir los permisos necesarios para el correcto funcionamiento de la aplicación, siendo sospechoso que se pidan algunos permisos, concretamente si son de los que la documentación oficial considera peligrosos, como los relacionados con las llamadas o SMS, por ejemplo.
- Información sobre la aplicación en si y los componentes que usa, incluyendo las pantallas o *activities* que se utilizan y el estilo o estilos que utilizan. También se puede cambiar en esta parte otros aspectos importantes como el icono o incluso el nombre de la aplicación. Anteriormente se indicaba en el archivo que nivel de API era el mínimo para el funcionamiento de la aplicación, pero eso se incluye ahora en el archivo de propiedades de Gradle.

- Otros datos importantes, como la key de una API. Para poder utilizar la API de Google Maps hay que solicitar una key, indicando luego en el archivo *manifest* cual es o la ruta hacia donde esté el valor almacenado.

Los componentes que controlan la vista y lógica de la aplicación serán, en nuestro caso, las actividades o *activities*. Las actividades gestionan el funcionamiento de la pantalla y la información que se maneja entre ellas. En este proyecto se encuentran 3 actividades importantes y una a modo de información:

- *MainActivity*: El funcionamiento es como el que se muestra en la siguiente figura, indicando que se necesita para el funcionamiento de dos de las actividades la activación del GPS.

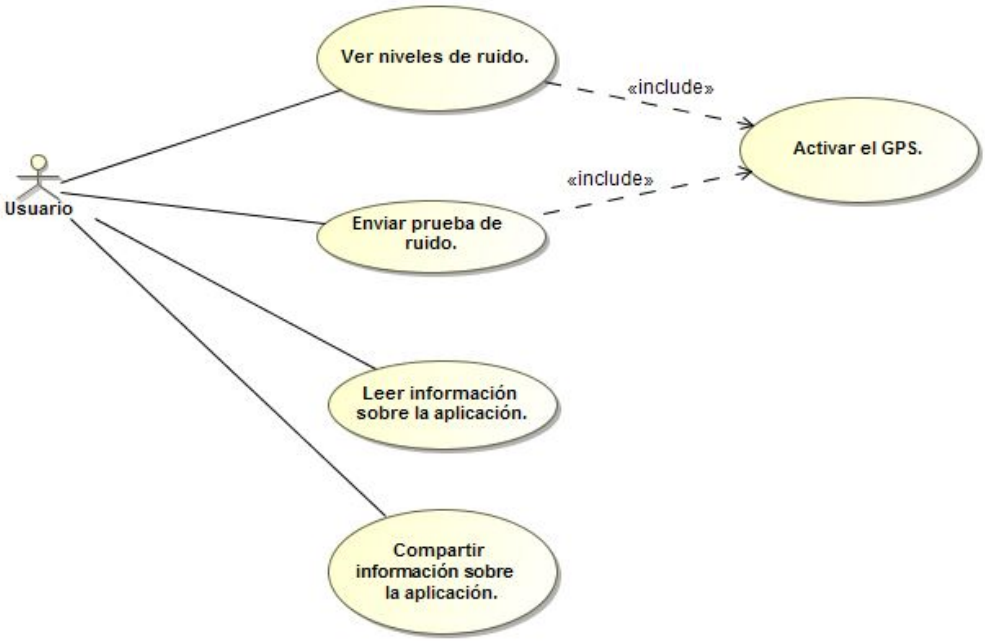

Figura 4.3: Caso de uso para *MainActivity*

- *NoiseActivity*: En esta actividad se puede ver un mapa con unos marcadores que indican los lugares de los cuales tiene datos el servidor. Puede seleccionarlos uno a uno los marcadores o puede darle al botón de siguiente, que se los enseñará uno a uno en bucle. También puede darle al botón de geolocalización para mostrarle donde se encuentra.

- *RecordActivity*: Esta actividad incluye una grabadora de sonidos, que toma la media del nivel de ruido que hay en el ambiente y envía dicha media al servidor. Para enviarlo se utiliza una clase que hereda de AsyncTask llamada *UploadTask*, enviando la localización también para que se muestre también en el mapa. El funcionamiento de esta actividad se describe en la siguiente figura:

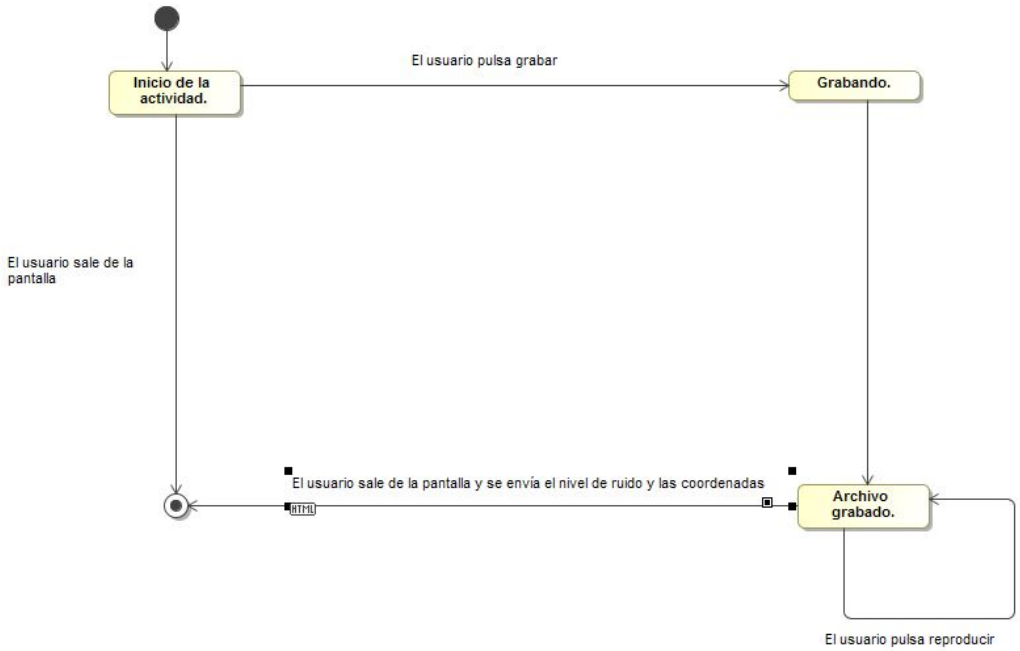

Figura 4.4: Diagrama de estados para *RecordActivity*

- *InformationActivity*: Una actividad para mostrar información mas resumida sobre el funcionamiento de la aplicación

La tercera y última parte importante de un proyecto en Android es la carpeta de recursos, que se organiza de la siguiente forma en este proyecto:

- *Drawable*: Aquí se incluyen archivos *bitmap* (.png, .9.png, .jpg, .gif) o archivos XML que son convertidos a *bitmap*.
- *layout*: Archivos XML para definir la interfaz de usuario. Dentro de estos archivos he usado, principalmente, los siguientes componentes:
    - *RelativeLayout*, que actúa principalmente como raíz del diseño y es donde se indican cuales son los elementos que forman parte de esa interfaz.
    - *LinearLayout*, para poner ordenados los elementos de la interfaz.
    - *TextView*, para enseñar textos.
    - *Buttons*, para cambiar de interfaz, parar la grabadora, etc.
    - *ImageView*, que son básicamente botones pequeños con una imagen y acción asociada.

        - *Fragment*, que se ha mencionado su funcionamiento antes y que también debe indicarse en el XML.

    - *Mipmap*: Otro tipo de imágenes.

    - *Values*: Aquí se almacenen distintos tipos de valores, organizándose así:

        - *Colors.xml* para los colores.
        - *Dimens.xml* para las dimensiones de algunos elementos.
        - *Strings.xml* para las cadenas de texto.
        - *Styles.xml* para diseñar estilos propios que se usarán en los botones y otros elementos.

Con la introducción de Gradle algunas de las propiedades que se indicaban en el *manifest* o otros archivos han pasado a incluirse en el archivo de Gradle de cada proyecto. Principalmente sirve para construir el archivo .apk y para hacer debugging al proyecto, en este archivo se incluye:

- Identificación de la aplicación, su código de versión, el nivel de API mínimo que utiliza y la versión del SDK de Android con el que se compila.

- El tipo de versión que se construye, pudiendo crear archivos .apk distintos para teléfonos y tabletas, por ejemplo.

- Las librerías a compilar. Por defecto se compila las librerías que se encuentren en la carpeta *libs* y JUnit pero también se pueden incluir otras librerías. Para este proyecto se han incluido las librerías de apoyo de Android (para hacer funcionar el proyecto en móviles con sistemas anteriores al del SDK con el que se compila) y la librería de Google Play Services, que es necesaria para poder utilizar la API de Google Maps.

Una queja que hay respecto a Gradle es que es bastante lento, algo que es verdad cuando hay que compilar muchas librerías externas, pero se puede mejorar la velocidad tocando unos pequeños ajustes de velocidad y memoria RAM que utiliza el compilador en el ordenador, pudiendo pasar a reducir los tiempos de compilación de manera drástica.

Como último toca mencionar la clase Java que hace las funciones de grabadora y de donde se saca el nivel de ruido. La clase, llamada SplEngine[1], tiene licencia GNU General Public License (GPL) versión 2, por lo cual puede ser utilizada perfectamente en este proyecto. Esta clase va recogiendo el nivel de ruido en un hilo aparte, que se comunica con la actividad que lo invocó a través de su *Handler*. Si bien su código es perfectamente funcional, tiene una componente respecto al audio que en la API actual se considera *deprecated*, es decir, no se recomienda su uso y otra respecto a la hora de guardar *logs* sobre el funcionamiento, aunque no he retocado esta al no usar esta función.

[1]creada por Hashir N.A, código fuente en https://code.google.com/archive/p/splmeter/

4.2.3. Explicación del funcionamiento de la integración.

La tarea de integración en Android es relativamente sencilla ya que para ello está la librería que comenté: Volley. Concretamente se hacen peticiones en 2 momentos:

- Cuando se inicia *NoiseActivity* se hace una petición GET al servidor, concretamente una de la forma GET /user-id/device-id/TO-DO , que es un archivo JSON que debe ser tratado para convertir su información (coordenadas y nivel de ruido) a marcador de Google Maps. La clase encargada de hacer la petición se llama *MarkerTask*.

- Cuando se acaba la actividad *RecordActivity* y se pasa de nuevo a MainActivity se hace una petición POST en la clase *UploadTask*. Este POST será un archivo JSON con la localización y el nivel de ruido.

Un ejemplo de archivo JSON que se maneja en el proyecto:

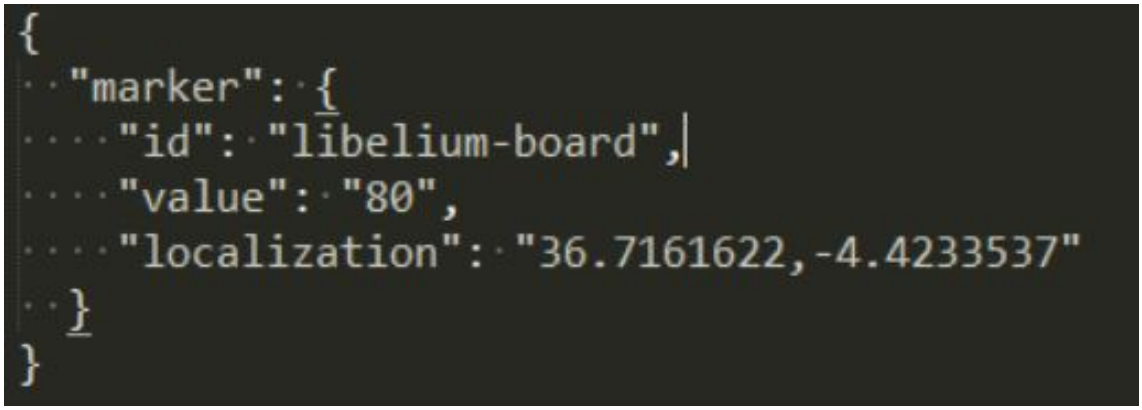

Figura 4.5: Ejemplo de JSON que se envía

4.2.4. Fases del desarrollo de la aplicación.

El desarrollo ha seguido un modelo basado en prototipos, que es un tipo de modelo de desarrollo evolutivo. Cada prototipo ha sido construido utilizando los componentes mínimos y necesarios hasta llegar al producto definitivo, es decir, a una aplicación completa y funcional.

Los prototipos que se han desarrollado son, en ese orden, los siguientes:

- Creación de la actividad principal, *MainActivity*, con el estilo deseado, incluyendo los botones hacia las correspondientes actividades (sin ningún funcionamiento asociado en el momento).

- Inclusión de una pantalla explicando el funcionamiento deseado de la aplicación y de un botón para compartir un mensaje sobre la aplicación.

- Creación de la actividad grabadora, *RecordActivity*, donde inicialmente solo grababa el audio. Se añade en este momento el permiso de *RECORD_AUDIO*, el de *ACCESS_FINE_LOCATION* y *ACCESS_COARSE_LOCATION*.

- Creación de la actividad del mapa de ruido, *NoiseActivity*, donde al principio solo estaba integrado el *fragment* del mapa con un marcador de prueba.
- Integración en el mapa de la clase que descarga los archivos JSON del servidor y su correspondiente conversión a marcadores de Google Maps. Se añade el permiso de *INTERNET*.
- Obtención del nivel de ruido en la actividad grabadora e integración del envío a través de la clase *UploadTask*.
- Integración de botón de geolocalización en la actividad del mapa y refinamiento del funcionamiento del GPS.

4.2.5. Código del proyecto.

Como el código de un proyecto Android es mas inmenso que el de la mota de Waspmote, solo voy a incluir los ficheros mas ligados a lo que es la programación, dejando por tanto fuera, dentro de la carpeta de recursos, lo que no sea los diseños en XML, ya que lo sobrante son cadenas de texto e imágenes. Al final de este apartado incluiré el retoque añadido a la clase que recoge el nivel de ruido, puesto que es la parte que he aportado yo a dicha clase, pudiendo encontrarse el enlace al repositorio de la clase original en la bibliografía. *AndroidManifest.xml:*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mangu.testing">
    <uses-permission android:name=
    "android.permission.RECORD_AUDIO"/>
    <uses-permission android:name=
    "android.permission.INTERNET"/>
    <uses-permission android:name=
    "android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name=
    "android.permission.ACCESS_COARSE_LOCATION"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name=
                "android.intent.action.MAIN"/>
```

```
                <category android:name=
                "android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".InformationActivity"/>
        <activity
            android:name=".RecordActivity"
            android:label=
            "@string/title_activity_record_activity"
            android:theme=
            "@style/AppTheme.NoActionBar" />
        <activity
            android:name=".NoiseActivity"
            android:label=
            "@string/title_activity_niveles"/>

        <meta-data
            android:name="com.google.android.geo.API_KEY"
            android:value="@string/google_maps_key" />


        <uses-library android:name="com.google.android.maps"/>

    </application>

</manifest>
```

build.gradle:

```
apply plugin: 'com.android.application'
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.3"
    defaultConfig {
        applicationId "com.mangu.tfg"
        minSdkVersion 16
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
```

```
            proguardFiles getDefaultProguardFile
            ('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.4.0'
    compile 'com.android.support:design:23.4.0'
    compile 'com.google.android.gms:play-services:8.4.0'
}
```

MainActivity:

```
public class MainActivity extends AppCompatActivity {
    private static final int UPLOAD_CODE = 101;
    private static final String TAG = "MainActivity";
    private static final String NPE_OR_INVALID_VALUE =
    "NPE or Invalid Value";
    protected LocationListener locationListener =
    new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            //mLocationManager.removeUpdates(locationListener);
        }
        @Override
        public void onProviderDisabled(String provider) {
        }
        @Override
        public void onProviderEnabled(String provider) {
        }
        @Override
        public void onStatusChanged(String provider,
        int status, Bundle extras) {
        }
    };
    private LocationManager mLocationManager;
    private RequestQueue mRequestQueue;
    private Intent openChooser;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mLocationManager = (LocationManager) getApplicationContext().
        getSystemService(LOCATION_SERVICE);
        mRequestQueue = Volley.newRequestQueue(
        getApplicationContext());
        ChooserTask chooserTask = new ChooserTask();
        chooserTask.execute();
    }

    public void onClickNoise(View view) {
        Intent intent = new Intent(getApplicationContext(),
        NoiseActivity.class);
        if (!mLocationManager.isProviderEnabled(
        LocationManager.GPS_PROVIDER)) {
            showAlert();
        }
        if (mLocationManager.isProviderEnabled(
        LocationManager.GPS_PROVIDER)) {
            startActivity(intent);
        }
    }
    public void onClickRecord(View view) {
        Intent intent = new Intent(getApplicationContext(),
        RecordActivity.class);
        if (!mLocationManager.isProviderEnabled(
        LocationManager.GPS_PROVIDER)) {
            showAlert();
        }
        if (mLocationManager.isProviderEnabled(
        LocationManager.GPS_PROVIDER)) {
            startActivityForResult(intent, UPLOAD_CODE);
        }
    }
    private void showAlert() {
        final AlertDialog.Builder builder =
        new AlertDialog.Builder(this);
        builder.setMessage("Debe activar el GPS para continuar")
                .setCancelable(false)
                .setPositiveButton("Activar",
                new DialogInterface.OnClickListener() {
                    public void onClick(
```

```
                @SuppressWarnings("unused")
                final DialogInterface dialog,
                @SuppressWarnings("unused") final int id) {
                    startActivity(new
                    Intent(android.provider.Settings.
                    ACTION_LOCATION_SOURCE_SETTINGS));
                }
            })
            .setNegativeButton("No activar", new
            DialogInterface.OnClickListener() {
                public void onClick(final DialogInterface dialog,
                 @SuppressWarnings("unused") final int id) {
                    dialog.cancel();
                }
            });
    final AlertDialog alert = builder.create();
    alert.show();
}

@Override
public void onActivityResult(int requestCode, int resultCode,
Intent data) {
    if (requestCode == UPLOAD_CODE) {
        if (resultCode == 1) {
            try {
                double value = data.getDoubleExtra(
                "value", -1);
                List<String> providers =
                mLocationManager.getProviders(true);
                Location bestLocation = null;
                for (String provider : providers) {
                    Location l = mLocationManager.
                    getLastKnownLocation(provider);
                    if (l == null) {
                        continue;
                    }
                    if (bestLocation == null ||
                    l.getAccuracy() < bestLocation.getAccuracy()) {
                        bestLocation = l;
                    }
                }
                if (bestLocation == null) {
```

```
                    Criteria criteria = new Criteria();
                    criteria.setAccuracy(Criteria
                    .ACCURACY_FINE);
                    criteria.setPowerRequirement(Criteria.
                    POWER_HIGH);
                    LocationListener lL = locationListener;
                    String provider = mLocationManager.
                    getBestProvider(criteria, true);
                    mLocationManager.
                    requestLocationUpdates(provider, 0, 0,
                    lL,
                    getMainLooper());
                    bestLocation =
                    mLocationManager.
                    getLastKnownLocation(
                    LocationManager.GPS_PROVIDER);
                }
                if (value != -1 && bestLocation != null) {
                    UploadTask uploadTask = new
                    UploadTask(getApplicationContext());
                    String lat_long = bestLocation.
                    getLatitude() + "," +
                    bestLocation.getLongitude();
                    uploadTask.execute(
                    String.valueOf(value), lat_long);
                } else {
                    Log.e(
                    NPE_OR_INVALID_VALUE, "Value:" + value);
                    if (bestLocation == null) {
                        Log.e(NPE_OR_INVALID_VALUE,
                        "BestLocation is null");
                    }
                }
                Log.i("onActivityResult",
                String.valueOf(value));
            } catch (SecurityException e) {
                Log.e("SecurityException",
                e.getLocalizedMessage());
            }
        }
    }
}
```

```
public void onClickInformation(View view) {
    Intent intent = new Intent(
    getApplicationContext(), InformationActivity.class);
    startActivity(intent);
}
public void onClickShare(View view) {
    if (this.openChooser != null) {
        startActivity(openChooser);
    } else {
        startActivity(generateIntent());
    }
}
public void setOpenChooser(Intent openChooser) {
    this.openChooser = openChooser;
}
public Intent generateIntent() {
    Resources resources = getResources();
    Intent emailIntent = new Intent();
    emailIntent.setAction(Intent.ACTION_SEND);
    emailIntent.putExtra(Intent.EXTRA_TEXT,
    Html.fromHtml(resources.getString(
    R.string.share_email_native)));
    emailIntent.putExtra(Intent.EXTRA_SUBJECT,
    resources.getString(R.string.share_email_subject));
    emailIntent.setType("message/rfc822");
    PackageManager pm = getPackageManager();
    Intent sendIntent = new Intent(Intent.ACTION_SEND);
    sendIntent.setType("text/plain");
    Intent openInChooser = Intent.createChooser(emailIntent,
    resources.getString(R.string.share_chooser_text));
    List<ResolveInfo> resInfo = pm.queryIntentActivities(
    sendIntent, 0);
    List<LabeledIntent> intentList = new ArrayList<>();
    for (int i = 0; i < resInfo.size(); i++) {
        ResolveInfo ri = resInfo.get(i);
        String packageName = ri.activityInfo.packageName;
        if (packageName.contains("android.email")) {
            emailIntent.setPackage(packageName);
        } else if (packageName.contains("twitter") ||
        packageName.contains("facebook") ||
         packageName.contains("mms") ||
        packageName.contains("android.gm")) {
```

```
                Intent intent = new Intent();
                intent.setComponent(new ComponentName(packageName,
                ri.activityInfo.name));
                intent.setAction(Intent.ACTION_SEND);
                intent.setType("text/plain");
                if (packageName.contains("twitter")) {
                    intent.putExtra(Intent.EXTRA_TEXT,
                    resources.getString(R.string.share_twitter));
                } else if (packageName.contains("facebook")) {
                    intent.putExtra(Intent.EXTRA_TEXT,
                    resources.getString(R.string.share_facebook));
                } else if (packageName.contains("android.gm")) {
                    intent.putExtra(Intent.EXTRA_TEXT,
                    Html.fromHtml(resources.getString(
                    R.string.share_email_gmail)));
                    intent.putExtra(Intent.EXTRA_SUBJECT,
                    resources.getString(
                    R.string.share_email_subject));
                    intent.setType("message/rfc822");
                }
                intentList.add(new
                LabeledIntent(intent, packageName,
                ri.loadLabel(pm), ri.icon));
            }
        }
        LabeledIntent[] extraIntents = intentList.toArray(new
        LabeledIntent[intentList.size()]);
        openInChooser.putExtra(
        Intent.EXTRA_INITIAL_INTENTS,
        extraIntents);
        return openInChooser;
    }

    private class UploadTask extends AsyncTask
    <String, Integer, Void> {
        private Context context;
        public UploadTask(Context context) {
            this.context = context;
        }
        @Override
        protected Void doInBackground(String... params) {
            JSONObject jsonObject = new JSONObject();
```

```
            JSONObject jsonArray = new JSONObject();
            JSONObject jsonRequest = new JSONObject();
            try {
                jsonObject.put("value", params[0]);
                jsonObject.put("localization", params[1]);
                jsonArray.put("marker", jsonObject);
                jsonRequest = new JSONObject
                (jsonArray.toString());
            } catch (JSONException e) {
                Log.e("JSONException", e.getMessage());
            } catch (NullPointerException e) {
                Log.e("NullPointerException", e.getMessage());
            }
            final String url = "http://150.214.108.91:8000";
            JsonObjectRequest req = new
            JsonObjectRequest(Request.Method.POST, url
                    , jsonRequest,
                    new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    try {

                VolleyLog.v("Response:%n %s",
                response.toString(4));
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }, new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {
                    VolleyLog.e("Error: ",
                    error.getMessage());
                }
            });
            req.setTag(TAG);
            mRequestQueue.add(req);
            return null;
        }
    }
    private class ChooserTask extends AsyncTask
    <Void, Void, Intent> {
```

```
        @Override
        protected Intent doInBackground(Void... params) {
            return generateIntent();
        }
        @Override
        protected void onPostExecute(Intent intent) {
            setOpenChooser(intent);
        }
    }
}
```

RecordActivity:

```
public class RecordActivity extends AppCompatActivity {
    private static final int MY_MSG = 1;
    private static final int ERROR_MSG = -1;
    private SplEngine mEngine;
    private Button start, stop, enviar;
    private boolean stopped = false;
    private double amplitude = -1; //Umbral
    public Handler mHandle = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what) {
                case MY_MSG:
                    amplitude = (double) msg.obj;
                    break;
                case ERROR_MSG:
                    break;
                default:
                    super.handleMessage(msg);
                    break;
            }
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_record);
        start = (Button) findViewById(R.id.btn_grabar);
        stop = (Button) findViewById(R.id.btn_prueba);
        enviar = (Button) findViewById(R.id.btn_enviar);
        stop.setEnabled(false);
```

```
        stop.getBackground().setColorFilter(Color.GRAY,
        PorterDuff.Mode.MULTIPLY);
        enviar.setEnabled(false);
        enviar.getBackground().setColorFilter(Color.GRAY,
        PorterDuff.Mode.MULTIPLY);
        mEngine = new SplEngine
        (this.mHandle, RecordActivity.this);
    }

    @Override
    public void onBackPressed() {
        finish();
    }

    public void start(View view) {
        if (!stopped) {
            mEngine.start_engine();
            start.setEnabled(false);
            start.getBackground().setColorFilter(Color.GRAY,
            PorterDuff.Mode.MULTIPLY);
            stop.setEnabled(true);
            Log.i(this.toString(), "Recording started");
        } else {
            Toast.makeText(RecordActivity.this, "Para grabar
            otro audio
            salga y vuelva a entrar en la pantalla",
            Toast.LENGTH_SHORT).show();
        }
    }

    public void stop(View view) {
        stopped = true;
        amplitude = mEngine.getMedian();
        mEngine.stop_engine();
        stop.setEnabled(false);
        stop.getBackground().setColorFilter(Color.GRAY,
        PorterDuff.Mode.MULTIPLY);
        start.setEnabled(true);
        start.getBackground().clearColorFilter();
        enviar.setEnabled(true);
        enviar.getBackground().clearColorFilter();
        Log.i(this.toString(), "Value: " + amplitude);
```

```
    }

    public void send(View view) {
        showAlert();
    }

    private void showAlert() {
        final AlertDialog.Builder builder = new
        AlertDialog.Builder(this);
        builder.setMessage("Nivel de ruido: "+ amplitude+ " dB")
        .setTitle("Enviar audio")
        .setCancelable(false)
                .setPositiveButton("Si", new
                DialogInterface.OnClickListener() {
                    public void onClick(
                    @SuppressWarnings("unused")
                    final DialogInterface dialog,
                    @SuppressWarnings("unused") final int id) {
                        Intent intent = new Intent();
                        intent.putExtra("value", amplitude);
                        setResult(1, intent);
                        finish();
                    }
                })
                .setNegativeButton("No", new
                DialogInterface.OnClickListener() {
                    public void onClick(
                    final DialogInterface dialog,
                    @SuppressWarnings("unused") final int id) {
                        dialog.cancel();
                    }
                });
        final AlertDialog alert = builder.create();
        alert.show();
    }
}
```

NoiseActivity:

```
public class NoiseActivity extends FragmentActivity
implements OnMapReadyCallback {
    private static final String TAG = "NoiseActivity";
    private GoogleMap mMap;
```

```
private LocationManager mLocationManager;
private List<Marker> mMarkerList;
private int mCounter = 0;
private List<String> mStringMarkerList;
private RequestQueue mRequestQueue;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mStringMarkerList = new ArrayList<>();
    mMarkerList = new ArrayList<>();
    MarkerTask markerTask = new MarkerTask
    (this.getApplicationContext());
    markerTask.execute();
    mRequestQueue = Volley.newRequestQueue
    (getApplicationContext());
    mLocationManager = (LocationManager)
    getApplicationContext().
    getSystemService(LOCATION_SERVICE);
    setContentView(R.layout.activity_noise);
    FragmentManager fManager = getSupportFragmentManager();
    Fragment fragment =
    fManager.findFragmentById(R.id.mapview);
    SupportMapFragment mapFragment =
    (SupportMapFragment) fragment;
    mapFragment.getMapAsync(this);
}
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    mMap.setOnMarkerClickListener(new
    GoogleMap.OnMarkerClickListener() {
        @Override
        public boolean onMarkerClick(Marker marker) {
            mMap.moveCamera(CameraUpdateFactory.
            newLatLngZoom(marker.getPosition(), 15));
            mMap.animateCamera(CameraUpdateFactory.zoomIn());
            mMap.animateCamera(CameraUpdateFactory.zoomTo(15)
            , 2000, null);
            marker.showInfoWindow();
            return false;
        }
    });
```

```
        for (String s : mStringMarkerList) {
            String[] splitted = s.split(";");
            String[] latlng = splitted[1].split(",");
            LatLng stringLatLng = new LatLng(
            Double.valueOf(latlng[0]),
            Double.valueOf(latlng[1]));
            Marker stringMarker = mMap.addMarker(new
            MarkerOptions().position(stringLatLng).
            title("Nivel de ruido:" +
            splitted[0]).icon(BitmapDescriptorFactory.
            fromResource(R.drawable.ic_image)));
            mMarkerList.add(stringMarker);
        }
        moveCamera(mMarkerList.get(0));
    }
    public void onClickFocus(View view) {
        try {
            List<String> providers = mLocationManager.
            getProviders(true);
            Location bestLocation = null;
            for (String provider : providers) {
                Location l = mLocationManager.
                getLastKnownLocation(provider);
                if (l == null) {
                    continue;
                }
                if (bestLocation == null ||
                l.getAccuracy() < bestLocation.
                getAccuracy()) {
                    bestLocation = l;
                }
            }
            if (bestLocation == null) {
                Criteria criteria = new Criteria();
                criteria.setAccuracy(Criteria.ACCURACY_FINE);
                criteria.setPowerRequirement(Criteria.
                POWER_HIGH);
                LocationListener lL = locationListener;
                String provider = mLocationManager.
                getBestProvider(criteria, true);
                mLocationManager.requestLocationUpdates(
                provider, 0, 0, lL,
```

```
                getMainLooper());
                bestLocation =
                mLocationManager.getLastKnownLocation(
                LocationManager.GPS_PROVIDER);
            }
            if (bestLocation != null) {
                LatLng newLatLng = new LatLng(
                bestLocation.getLatitude(),
                bestLocation.getLongitude());
                Marker actualLocation = mMap.addMarker(new
                MarkerOptions().position(newLatLng).title
                ("Usted esta
                aqui").snippet("Ultima localizacion
                conocida").icon(
                BitmapDescriptorFactory.fromResource(
                R.drawable.ic_localizar)));
                actualLocation.showInfoWindow();
                mMap.moveCamera(
                CameraUpdateFactory.newLatLng(newLatLng));
            } else {
                Toast.makeText(NoiseActivity.this, "
                Su dispositivo no tiene localizaciones en cache",
                Toast.LENGTH_SHORT).show();
                Log.e("NoiseActivity",
                "No hay localizaciones en cache");
            }
        } catch (SecurityException e) {
            Log.e("SecurityException", e.getLocalizedMessage());
        } catch (NullPointerException ex) {
            Log.e("NullPointerException",
            ex.getLocalizedMessage());
        }
    }
    public void onClickNext(View view) {
        int next = mCounter % mMarkerList.size();
        if (next < mMarkerList.size()) {
            moveCamera(next);
            mCounter++;
        }
    }
    public void moveCamera(int next) {
        Marker next_marker = mMarkerList.get(next);
```

```
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(
        next_marker.getPosition(), 15));
        mMap.animateCamera(CameraUpdateFactory.zoomIn());
        mMap.animateCamera(CameraUpdateFactory.zoomTo(15),
         2000, null);
        next_marker.showInfoWindow();
    }
    public void moveCamera(Marker marker) {
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(
        marker.getPosition(), 15));
        mMap.animateCamera(CameraUpdateFactory.zoomIn());
        mMap.animateCamera(CameraUpdateFactory.zoomTo(15),
         2000, null);
        marker.showInfoWindow();
    }
    private class MarkerTask extends AsyncTask
    <Void, Integer, Void> {
        Context context;
        public MarkerTask(Context context) {
            this.context = context;
        }
        @Override
        protected Void doInBackground(Void... params) {
            final String url = "http://150.214.108.91:8000";
            JsonArrayRequest req = new JsonArrayRequest(url,
                    new Response.Listener<JSONArray>() {
                        @Override
                        public void onResponse(
                        JSONArray response) {
                            try {
                                for(int i =0; i<response.
                                length();i++) {
                                    JSONObject marker =
                                    (JSONObject) response.get(i);
                                    String toAdd =
                                    marker.getString("value")
                                    +";"+marker.getString
                                    ("localization");
                                    mStringMarkerList.add(toAdd);
                                }
                            } catch (JSONException e) {
                                Log.e(e.toString(),
```

```
                                e.getMessage());
                        }
                    }
                }, new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {
                    VolleyLog.e("Error: ", error.getMessage());
                }
            });
            req.setTag(TAG);
            mRequestQueue.add(req);
            return null;
        }
    }
    protected LocationListener locationListener =
    new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            //mLocationManager.removeUpdates(locationListener);
        }
        @Override
        public void onProviderDisabled(String provider) {
        }
        @Override
        public void onProviderEnabled(String provider) {
        }
        @Override
        public void onStatusChanged(String provider,
        int status, Bundle extras) {
        }
    };
}
```

InformationActivity:

```
public class InformationActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_information);
    }
    public void onClickMail(View view) {
        Intent intent = new Intent(Intent.ACTION_SEND);
```

```
        intent.setType("plain/text");
        intent.putExtra(Intent.EXTRA_EMAIL, new String[]
        {this.getString(R.string.correo)});
        intent.putExtra(Intent.EXTRA_SUBJECT,"Sobre
        aplicacion TFG");
        startActivity(Intent.createChooser(intent, ""));
    }
}
```

activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance=
        "?android:attr/textAppearanceLarge"
        android:text="@string/Bienvenidos"
        android:id="@+id/textView1"
     />
    <View
        android:layout_width="fill_parent"
        android:layout_height="5dp"
        android:background="#ffffff"/>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1.0"
        android:orientation="horizontal" >
        <Button
            android:id="@+id/btn_niveles"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            style="@style/btnStyleAcapulco"
            android:layout_weight="1.0"
            android:text="@string/Niveles"
            android:onClick="onClickNoise"
```

```
            />
        <Button
            android:id="@+id/btn_prueba"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="1.0"
            style="@style/btnStyleAcapulco"
            android:text="@string/Subirniveles"
            android:onClick="onClickRecord"
            />
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1.0"
        android:orientation="horizontal" >
        <Button
            android:id="@+id/btn_informacion"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            style="@style/btnStyleAcapulco"
            android:layout_weight="1.0"
            android:text="@string/Informacion"
            android:onClick="onClickInformation"
            />
        <Button
            android:id="@+id/btn_compartir"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            style="@style/btnStyleAcapulco"
            android:layout_weight="1.0"
            android:text="@string/Compartir"
            android:onClick="onClickShare"
            />
    </LinearLayout>
</LinearLayout>
```

activity_record.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">
    <ImageView
        android:id="@+id/imageView1"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:contentDescription="@string/microfono"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="37dp"
        android:scaleType="fitXY"
        android:src="@drawable/ic_image" />
    <Button
        android:id="@+id/btn_grabar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imageView1"
        android:layout_marginTop="67dp"
        android:layout_toLeftOf="@+id/imageView1"
        android:onClick="start"
        android:contentDescription="@string/microfono"
        android:text="@string/start"
        />
    <Button
        android:id="@+id/btn_prueba"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="stop"
        android:text="@string/stop"
        android:layout_alignTop="@+id/btn_grabar"
        android:layout_toRightOf="@+id/imageView1"
        android:layout_toEndOf="@+id/imageView1" />
    <Button
        android:id="@+id/btn_enviar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/btn_prueba"
        android:layout_centerHorizontal="true"
        android:onClick="send"
        android:text="@string/enviar" />
```

```
</RelativeLayout>
```

activity_noise.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:id="@+id/map"
    tools:context=".NoiseActivity"
    android:name=
    "com.google.android.gms.maps.SupportMapFragment">
    <fragment
        android:id="@+id/mapview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class=
        "com.google.android.gms.maps.SupportMapFragment"
        >
        <ImageButton android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="right|top"
            android:src="@drawable/ic_localizar"
            android:contentDescription="@string/mi_localizacion"
            android:background=
            "?android:selectableItemBackground"
            android:padding="10dp"
            android:layout_marginTop="10dp"
            android:paddingRight="10dp"
            android:onClick="onClickFocus"
            />
        <ImageButton android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="left|top"
            android:background=
            "?android:selectableItemBackground"
            android:contentDescription="@string/siguiente"
            android:src="@mipmap/ic_next"
            android:padding="10dp"
            android:layout_marginTop="10dp"
```

```
            android:paddingRight="10dp"
            android:onClick="onClickNext"
            />
        </fragment>
</LinearLayout>
```

activity_information.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".InformationActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance=
        "?android:attr/textAppearanceLarge"
        android:text="@string/Seccion_Ayuda"
        android:id="@+id/tv_info"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance=
        "?android:attr/textAppearanceLarge"
        android:text="@string/correo"
        android:id="@+id/tv_correo"
        android:layout_below="@+id/tv_info"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:onClick="onClickMail"
        />
</RelativeLayout>
```

Capítulo 5

Conclusiones y trabajo futuro.

5.1. Conclusiones

El desarrollo de un trabajo de fin de grado con las características de este da para muchas conclusiones y anécdotas para futuros trabajos. Algunas de estas conclusiones están mas relacionadas con mi interés personal a la hora de hacer desarrollos y otras con cuestiones mas técnicas.

5.1.1. Mi opinión sobre el desarrollo en Android

Aunque previamente ya había hecho algunos pequeños desarrollos en Android, esta ha sido la primera vez que me he enfrentado a un desarrollo mas importante para la plataforma.

Teniendo claro desde el momento en el que escribí el anteproyecto que, dados mis conocimientos en ese momento, apenas conocía que herramientas y métodos podrían serme útiles, empecé a formarme en partes mas avanzadas del desarrollo en Android: manejo mas avanzado de *threads*, el uso de una nueva API para las conexiones a Internet (antes había utilizado la de Apache), etcétera.

Otra parte importante del desarrollo surgió a mitad del mismo. En un principio yo había considerado la opción de que, a la hora de subir un audio al servidor, el usuario también pudiese escoger un audio previamente grabado. El problema que encontré fue que no había manera viable de poder sacar la media del nivel de ruido dentro de la aplicación en un audio grabado sin cargar mas horas al proyecto o tener que utilizar software privativo.

El balance al finalizar el proyecto es bastante positivo, puesto que he utilizado conceptos del desarrollo de Android bastante interesantes de cara al mercado laboral, pero también ha sido interesante para ganar conocimientos sobre este mundo.

Uno de los problemas relacionados con este desarrollo fue a la hora de *debuggear* la aplicación. Para hacer esto hay dos principales opciones: sobre un dispositivo físico y

sobre una máquina virtual. Para otras aplicaciones personalmente habría trabajado con una máquina virtual de Genymotion, que corre sobre Oracle VirtualBox, sin embargo, no permite el uso de micrófonos, lo cual hace imposible que pruebe el 100 % de mi aplicación de esa forma, teniendo que escoger por tanto probar en un dispositivo físico, concretamente en mi teléfono con Android 5.1.

5.1.2. Mi opinión sobre el desarrollo para Waspmote

Los dos principales problemas que me he encontrado a la hora de desarrollar el *sketch* han sido:

- La falta de soporte para las placas, que solo tienen soporte en el foro oficial, donde rara vez la ayuda que se da por parte de los moderadores es útil. Ante cualquier obstáculo u problema que me encontrase durante el desarrollo, las primeras soluciones siempre eran relacionadas con Arduino y dichos problemas habían sido ya subsanados en esa plataforma.

- La versión del compilador de la placa es bastante anticuada, provocando que los *sketchs* tarden en compilar bastante, lo cual hace que se pierda mas tiempo porque solo detecta un fallo por cada compilación y no varios.

Tras resolver dichos problemas, he podido ver que el desarrollo para estas motas es bastante mas rápido que para Arduino y mucho mas liviano, teniendo en cuenta cosas como que el tamaño de los añadidos para conectividad WiFi en Arduino es un *shield* casi mas grande que el propio Arduino y en Waspmote es un pequeño añadido que va en una esquina.

La programación para la mota también sufre al tener una versión antigua del compilador, como comenté anteriormente sobre el uso de una función. Sin embargo, no se necesitaría tener conocimientos previos de C en la mayoría de los casos de la mota, puesto que la bibliografía ayuda mucho a la hora de crear los *sketchs*.

5.1.3. Análisis de los resultados

Se han cubierto las siguientes funcionalidades de las propuestas en el anteproyecto:

- Despliegue de sensores que recojan los niveles de ruido y los envíen a un servidor.

- Desarrollo de una aplicación para dispositivos Android que pueda realizar:

    - Ver los niveles de ruido de las zonas donde hay información.
    - Poder enviar información sobre el nivel de ruido de una zona.

Se han añadido las siguientes funcionalidades extra:

- Poder ver los puntos sobre los que hay información uno detrás de otro, como una lista.
- Botón para compartir con aplicaciones de redes sociales y mensajería instantánea que se está usando la aplicación.

Las siguientes funcionalidades no se han podido cubrir:

- Usar el dispositivo del usuario a modo de pasarela para las placas, que al final utilizan una conexión WiFi propia por dificultades para la integración con dispositivos Android que se explican mas adelante.
- Ver histórico de los datos, que no se ha implementado debido a que, a medida que avanzaba el desarrollo de la aplicación, no le veía interés a que el usuario final pudiese ver el histórico de datos, puesto que es algo que no le es útil.

La aplicación se comporta como se espera: los mensajes llegan al servidor perfectamente y sin ningún problema relacionado con los puertos o la conexión. Hubo que realizar una calibración en la grabadora para que comenzara a grabar desde justo el momento que se le da a grabar, puesto que si no se hacía dicha calibración podría no recoger información y dar como resultado un valor nulo.

5.2. Trabajo futuro.

Si este desarrollo se ampliase o modificase en un futuro, sería conveniente tener en cuenta los siguientes aspectos:

- Portar todo el desarrollo, o parte del mismo, a Kotlin: Kotlin es el futuro para el desarrollo de aplicaciones en Android, dandose incluso el caso de que Google lo está considerando, junto a otros lenguajes, como sustituto de Java. Poco a poco, son muchas las personas que van descubriendo las bondades de Kotlin, como por ejemplo:
  - Conciso: Se necesitan muchas menos líneas de código para muchas tareas, comparandolo con Java. Por ejemplo, para conseguir la cantidad de números positivos en una lista, en Kotlin se haría tal que así:

```
val positiveNumbers = list.filter {it > 0}
```

    o para hacer una *AsyncTask*, con la librería de Anko, que suele acompañar muchos desarrollos en Kotlin, queda así:

```
async() {
    // Do something in a secondary thread
    uiThread {
```

```
            // Back to the main thread
        }
    }
```

- Programación funcional: Si bien la última versión de Android, Android N, si tiene las funciones lambda, con Kotlin es posible incluirlas, junto con todo el paradigma de la programación funcional, para las versiones anteriores, siendo algo que ayuda mucho a la hora de escribir menos código.
- Seguro: Se pueden evitar muchísimos tipos de errores, sobretodo los relacionados con las referencias nulas. Por ejemplo, con el uso del operador ?:, conocido como el operador Elvis, podemos hacer cosas así, que evitan las referencias nulas.

```
//maybe es un String que puede ser nulo
val name : String = maybe ?: "stranger"
println("Hello $name")
```

- Versatilidad: Kotlin, aparte de en Android, puede funcionar principalmente en aplicaciones del lado del servidor, como es el caso de la web de presentaciones Prezi, o en el *front-end* de un navegador, haciendo que lo que se aprenda de Kotlin no se considere del todo inútil al cambiar de área de desarrollo.
- Interoperabilidad: Funciona perfectamente con la máquina virtual de Java y con otros *frameworks*.

- Portar la aplicación a Android Wear: Pensado para casi cualquier tipo de *wearables*, Android Wear es un sistema operativo basado en Android que responde a las necesidades y limitaciones de estos dispositivos. Aunque por ahora no hay muchos dispositivos a la venta de esta familia, podría ser interesante el desarrollo para estos dispositivos. No habría que cambiar mucho en la aplicación, tan solo adaptar temas de resolución, puesto que todo el código utilizado en la aplicación de este proyecto es compatible en Android Wear, teniendo que incluir tan solo en el archivo Gradle la linea de las librerías de Android Wear.

- Mejorar la experiencia en tablets: Si bien la aplicación funciona perfectamente en las mismas, estaría bien retocar un poco las vistas para resoluciones mas propias de tablets, pudiendo adoptarlas mejor a su funcionamiento.

- Portar la aplicación a iOS: Si bien con Android cubrimos la mayor parte del mercado de dispositivos móviles como se comentó anteriormente, portando la aplicación a iOS terminaríamos de cubrir el mercado, quedando una parte casi despreciable sin cubrir. Para poder desarrollar aplicaciones para el sistema necesitamos principalmente unirnos al *Apple Developer Program*, donde se nos

indicarán todos los pasos para poder lanzar las aplicaciones, pudiendo realizar el desarrollo en Objective-C, Swift, o algún lenguaje multiplataforma.

- Portar el código a Xamarin: En la linea de lo mencionado anteriormente, Xamarin permitiría sacar la aplicación para Android, iOS y Windows Mobile partiendo del mismo código. En este supuesto nos ahorraríamos de mantener 2 o 3 códigos distintos, ahorrando un tiempo que podría valer en introducir optimizaciones del funcionamiento general.

- Utilizar la información recogida: Si bien la información recogida se almacena en un servidor que yo no he desarrollado, si considero interesante que esta información se pueda utilizar para mas propósitos. Algo interesante para lo que se podrían usar estos datos es el *Big Data*. El *Big Data* hace referencia a una cantidad de datos tal que supera la capacidad del software tradicional para ser capturados, administrados y procesados. Si bien al principio los datos que se nos presentan no se acercan a esta descripción, con el paso del tiempo se podría acabar generar tal volumen de información que sería interesante utilizar las técnicas mas asociadas al *Big Data* para analizar dicha información. En un supuesto de desplegar este proyecto en sitios de la ciudad, se podría acabar estudiando las zonas mas ruidosas y dando estadísticas de ruido que puedan ayudar a la Administración Pública, por ejemplo, si hubiera problemas con los vecinos en esas zonas.

Capítulo 6

Apéndice.

6.1. Lista de acrónimos

API: Application Programming Interface
EEPROM: Electrically Erasable Programmable Read-Only Memory
ext4: fourth extended filesystem
GPL: General Public License
GPS: Global Positioning System
HTTP: Hypertext Transfer Protocol
IDE: Integrated Development Environment
JIT: Just In Time
JSON: JavaScript Object Notation
NFC: Near Field Communication
REST: Representational State Transfer
RTC: Real Time Clock
SRAM: Static Random Access Memory
WEP: Wired Equivalent Privacy
WPA: Wi-Fi Protected Access
XML: eXtensible Markup Language
YAFFS: Yet Another Flash File System

6.2. Manual de usuario y capturas de pantalla.

6.2.1. Aplicación para dispositivos Android.

La aplicación no está subida en Play Google, así que para instalarla es necesario tener acceso al archivo .apk de la misma. Para poder instalarla es necesario activar la opción de instalar aplicaciones de orígenes desconocidos, que se encuentra en el apartado de seguridad de los ajustes. En el mapa podemos ver los puntos sobre los que hay información.

También podemos pulsar en el botón de la esquina superior izquierda para ir al siguiente punto del que haya información. Con el botón de la esquina superior derecha te enseña donde te encuentras, siempre y cuando el GPS tenga información actualizada.

Figura 6.1: Punto donde se han realizado pruebas de sonido.

Figura 6.2: Localización actual.

Esta es la pantalla de la grabadora, con un funcionamiento ya explicado anteriormente.

Figura 6.3: Estado inicial de la grabadora.

Figura 6.4: Grabando.

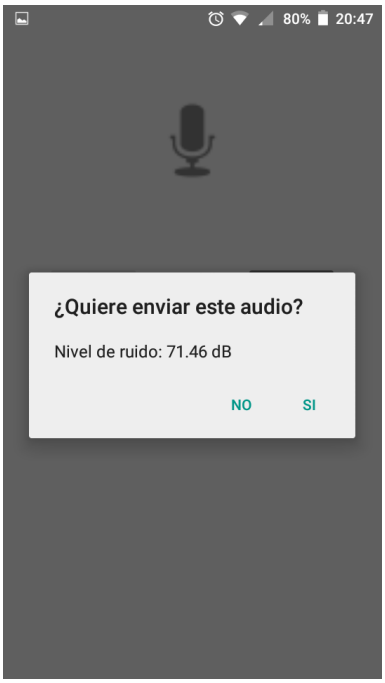

Figura 6.5: Prueba de envío de audio.

Cuando se pulsa el botón de compartir se puede elegir ciertas aplicaciones con las que compartir que se está usando la aplicación.

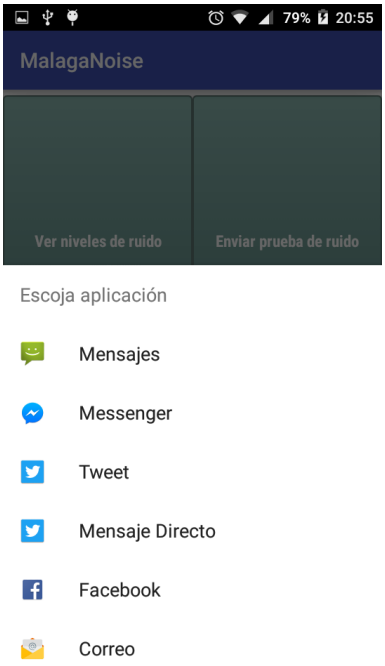

Figura 6.6: Escogiendo que aplicación usar para compartir información

6.2.2. *Sketch* para Waspmote.

Para poder utilizar la placa en Windows no es necesario instalar ningún driver especial. Para poder lanzar el código es necesario bajarse el IDE propio de Libelium, pudiendo dar problemas si se usa otro. Primero debe compilarse el código y luego cargarse a la placa.

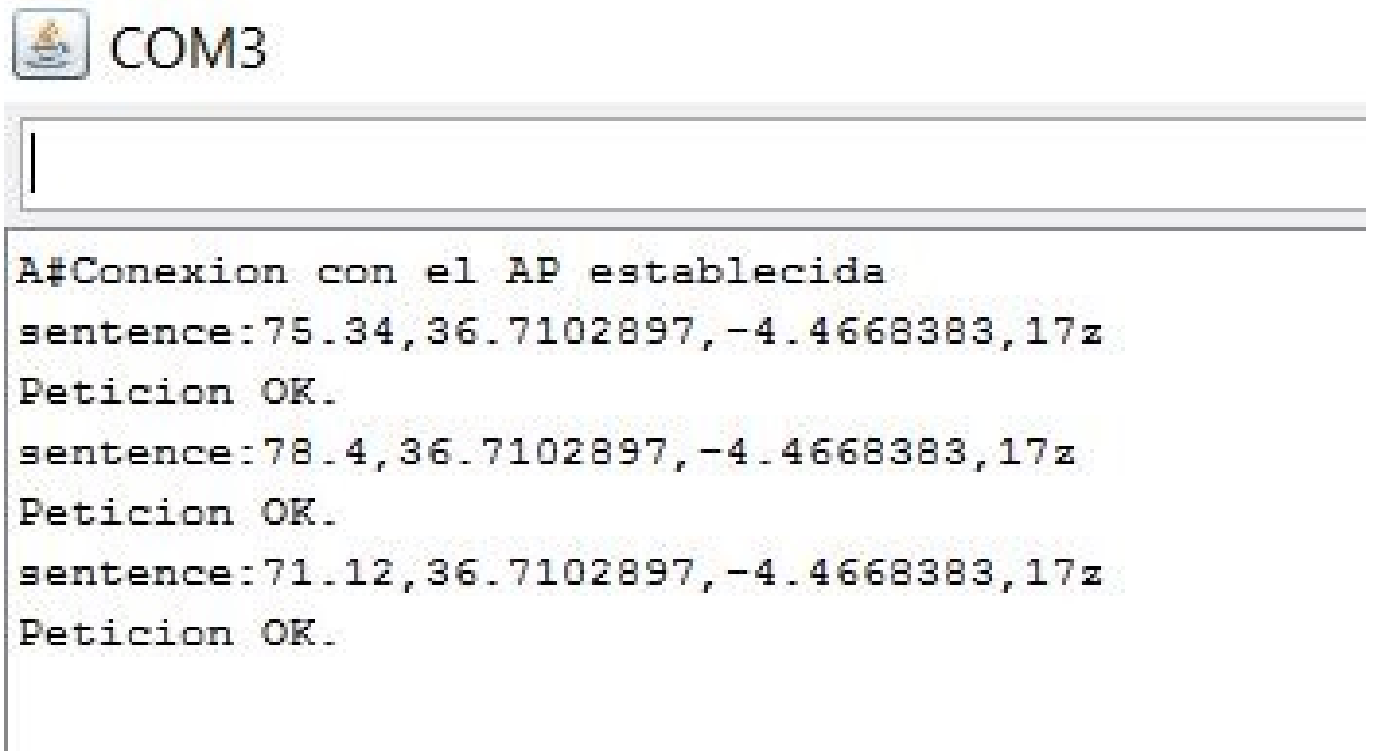

Figura 6.7: Muestra del funcionamiento de la placa

Bibliografía

[1] International Data Corporation. "Smartphone OS Market Share, 2015 Q2". En: *RFID Journal* (2015).

[2] K Ashton. "That "Internet of things" thing". En: *RFID Journal* (2009).

[3] R Hollands. "Will the real smart city please stand up?" En: *City: analysis of urban trends, culture, theory, policy, action* (2008), págs. 303-320.

[4] Ben Elgin. "Google Buys Android for Its Mobile Arsenal". En: (2005).

[5] Google. "Android Marshmallow Release Notes". En: (2016).

[6] Agencia Kantar. "Cuota de mercado de smartphones en España". En: (2016).

[7] Statista. "Share of Android plataforms". En: (2016).

[8] Mike Gouline. "Kotlin, the Swift of Android". En: (2014).

[9] Google. "Android Studio, the official IDE for Android". En: (2016).

[10] Jesús Tomás Girones. "El Gran libro de Android". En: (2016).

[11] AppBrain. "Number of available Android apps". En: (2016).

[12] Google. "Centro de Políticas de Desarrolladores". En: (2016).

[13] Libelium. "Waspmote Product Description". En: (2016).

[14] Libelium. "Waspmote Technical Guide". En: (2016).